SSE 643 - Advanced Graphics

Project 1

Dr. Paul MacNeil

2/15/10

By: Tim Barnes

Contents

[Project Goals 3](#_Toc254011877)

[Part 1: Advanced Graphic Interfaces 3](#_Toc254011878)

[Part 2: Wiki 3](#_Toc254011879)

[Part 1: XNA Game Studio 4](#_Toc254011880)

[Getting Started – My First XNA Game 4](#_Toc254011881)

[Semester Project – OTV Miner 5](#_Toc254011882)

[Project 1 – Asteroid Mining 5](#_Toc254011883)

[Goals 5](#_Toc254011884)

[GameObject Class 5](#_Toc254011885)

[Displaying Objects 6](#_Toc254011886)

[Asteroid Movement 9](#_Toc254011887)

[Asteroid Breakup 10](#_Toc254011888)

[Element Release 12](#_Toc254011889)

[Target Hitting 12](#_Toc254011890)

[Gameplay 14](#_Toc254011891)

[Conclusion 17](#_Toc254011892)

[Part 2: Wiki 17](#_Toc254011893)

[Main Page 18](#_Toc254011894)

[SSE 643 Advanced Graphics Interfaces Page 19](#_Toc254011895)

[Project 1 Page 19](#_Toc254011896)

[Cleaning the Navigation Bar 20](#_Toc254011897)

# Project Goals

## Part 1: Advanced Graphic Interfaces

Demonstrate capabilities in topics covered in the XNA Game Studio Documentation.

## Part 2: Wiki

Learn to create and edit a Wiki page for showing reports of projects.

# Part 1: XNA Game Studio

## Getting Started – My First XNA Game

After installing the XNA Framework 3.1 plug-in for Visual Studio 2008, I began working through the first tutorial online called “Beginner’s Guide to 2D Games,” since I knew nothing about the XNA framework. This tutorial creates a simple 2D game that consists of a cannon firing cannon balls at alien space ships moving across the screen. Below is image taken from the game after I completed the tutorial.



I modified some of the aspects of the tutorial in order to extend my knowledge and to figure out how each of the variables interact with the XNA framework. I moved the cannon from the left side of the screen to middle, while adding to range of motion. I also added more cannon balls and alien space ships. This tutorial was quick and simple, but was essential for me learning how the XNA framework works and what it takes to get objects displaying and updating on the screen.

## Semester Project – OTV Miner

In the beginning of the semester, I teamed up with Les Laurell from the course ListServ. We had a meeting and came up with a semester project that would produce a full game. During our meeting we divided up the labor needed to accomplish this task, while giving each a chance to learn the XNA framework and use our own creativity for each of the individual projects for the course (Project 1 & Project 2). We will combine each project together for Project 3 and add some final touches to make a full working game.

The OTV Miner will be space game that has ship mining asteroids for fuel and elements, while avoiding or fighting foreign ships that are after the same fuel and elements. The ship will need to fire a laser at asteroids in order to mine their contents. The contents give the ship fuel for movement and to continue using the laser. Also during mining, the element Osmium (Os3) can be released and collected by the ship. Collecting enough Os3 is how you beat the game.

## Project 1 – Asteroid Mining

Les and I are working independently on this project but our work will be combined later in order to build the full game.

### Goals

My goals for this project were:

* Asteroid Movement
* Asteroid Breakup
* Element Release
* Target Hitting

### GameObject Class

The game GameObject class is a based class for all sprites displayed in the game. All core variables needed to display, move, and destroy any object is contained in this base class.

class GameObject

{

private Texture2D sprite;

private Vector2 position;

private float rotation;

private Vector2 center;

private Vector2 velocity;

private bool alive;

public GameObject(Texture2D loadedTexture)

{

rotation = 0.0f;

position = Vector2.Zero;

sprite = loadedTexture;

center = new Vector2(sprite.Width / 2, sprite.Height / 2);

velocity = Vector2.Zero;

}

public Texture2D Sprite

{

get { return sprite; }

set {

this.sprite = value;

center = new Vector2(sprite.Width / 2, sprite.Height / 2);

}

}

public Vector2 Position

{

get { return position; }

set { this.position = value; }

}

public float XPos

{

get { return position.X; }

set { this.position.X = value; }

}

public float YPos

{

get { return position.Y; }

set { this.position.Y = value; }

}

public Vector2 Center

{

get { return center; }

set { this.center = value; }

}

public Vector2 Velocity

{

get { return velocity; }

set { this.velocity = value; }

}

public float Rotation

{

get { return rotation; }

set { this.rotation = value; }

}

public bool IsAlive

{

get { return alive; }

set { this.alive = value; }

}

}

A GameObject is created by calling the constructor and passing it a Texture2D object, which contains the image from the Contents section in the XNA framework. Once called, the constructor sets up all the variables and calculates the center of the Texture2D object. I added XPos & YPos to the Properties (Get\Set) for ease of use when manipulating the GameObject.

### Displaying Objects

There three main functions in Game1.cs that need to be edited to display any object to the screen. The Game1 class is main class of the program that the XNA framework using to continually update the game. The three methods are LoadContent, Update, and Draw.

#### LoadContent Method

The LoadContent method is simply a function that is used to allow the programmer the ability to initialize any variable and objects that will be needed for the game. This method is called when the game is loaded.

protected override void LoadContent()

{

// Create a new SpriteBatch, which can be used to draw textures.

spriteBatch = new SpriteBatch(GraphicsDevice);

theShip = new GameObject(Content.Load<Texture2D>("Sprites\\ship2"));

theShip.Position = new Vector2(graphics.GraphicsDevice.Viewport.Width/2,

graphics.GraphicsDevice.Viewport.Height -100);

lasers = new List<GameObject>();

for (int i = 0; i < maxLasers; i++)

{

lasers.Add(new GameObject(

Content.Load<Texture2D>("Sprites\\laser")));

}

asteroids = new List<AsteroidObject>();

for (int i = 0; i < maxAsteroids; i++)

{

asteroids.Add(new AsteroidObject(GetRandomLargeAsteroidContent()));

}

elements = new List<GameObject>();

viewportRect = new Rectangle(0, 0,

graphics.GraphicsDevice.Viewport.Width,

graphics.GraphicsDevice.Viewport.Height);

}

I used this method to load the ship, lasers, and asteroids to the game. I also calculated the viewportRect which is a rectangle that holds the dimensions of the viewable screen.

#### Update Method

The Update method is called repeatedly throughout the gameplay. It is basically called on a time interval by the XNA framework and is the place code where all the objects that need to be manipulated happens.

protected override void Update(GameTime gameTime)

{

// Allows the game to exit

if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)

this.Exit();

KeyboardState keyboardState = Keyboard.GetState();

if (keyboardState.IsKeyDown(Keys.Left))

{

theShip.XPos -= 5;

}

if (keyboardState.IsKeyDown(Keys.Right))

{

theShip.XPos += 5;

}

if (keyboardState.IsKeyDown(Keys.Space) &&

prevKeyboardState.IsKeyUp(Keys.Space))

{

FireLasers();

}

UpdateLasers();

UpdateAsteroids();

prevKeyboardState = keyboardState;

base.Update(gameTime);

}

The Update method is also where I handled all user inputs to control the ship and fire the lasers. Checking the keyboard state allows me to move the ship left and right and also allows the lasers to be fired. The keyboard state also makes it possible to only allow one laser to be fire per press of the spacebar, without it, it would fire lasers even time the Update method is called and the spacebar is down.

The methods FireLasers, UpdateLasers, and UpdateAsteroids handles all the logic that updates all the properties of each object based on current conditions in the game. These methods will be seen later in this report.

#### Draw Method

The Draw method is called immediately after the Update method in order to redraw all the contents of the screen. The basic functionality of this method is to go through all of the collections of objects to be displayed and redraw their sprite based on its need position, which may have got updated in the Update method.

protected override void Draw(GameTime gameTime)

{

GraphicsDevice.Clear(Color.Black);

spriteBatch.Begin(SpriteBlendMode.AlphaBlend);

foreach (GameObject laser in lasers)

{

if (laser.IsAlive)

{

spriteBatch.Draw(laser.Sprite, laser.Position, Color.White);

}

}

spriteBatch.Draw(theShip.Sprite, theShip.Position, null, Color.White);

foreach (AsteroidObject asteroid in asteroids)

{

if (asteroid.IsAlive)

spriteBatch.Draw(asteroid.Sprite, asteroid.Position, Color.White);

}

foreach (GameObject element in elements)

{

if (element.IsAlive)

spriteBatch.Draw(element.Sprite, element.Position, Color.White);

}

spriteBatch.End();

base.Draw(gameTime);

}

The main constraint on whether to draw an object or not is based on whether it is alive or not. This is determined in the Update on whether it has collided with another object or if it has moved off the screen.

### Asteroid Movement

The asteroids in the game will be roaming around the map randomly and the ship will need to find the asteroids in order to mine them for their contents. The UpdateAsteroids method, in the Update function from Game1.cs, is use to handle the movement of the asteroids.

private void UpdateAsteroids()

{

foreach (AsteroidObject asteroid in asteroids)

{

if (asteroid.IsAlive)

{

asteroid.Position += asteroid.Velocity;

if(!viewportRect.Contains(new Point((int)asteroid.XPos,

(int)asteroid.YPos)))

{

asteroid.IsAlive = false;

continue;

}

}

else

{

asteroid.IsAlive = true;

asteroid.AsteroidState = AsteroidState.Large;

asteroid.Sprite = GetRandomLargeAsteroidContent();

asteroid.Position = new Vector2(

MathHelper.Lerp((float)viewportRect.Left +

asteroid.Sprite.Width,

(float)viewportRect.Right - asteroid.Sprite.Width,

(float)random.NextDouble()),

viewportRect.Top);

asteroid.Velocity = new Vector2(0,

MathHelper.Lerp(minAsteriodVelocity,

maxAsteriodVelocity,

(float)random.NextDouble()));

}

}

}

The UpdateAsteroids() method goes through each asteroid in a list of asteroids to determine if they are alive or not. If they are alive, the asteroid position is updated by adding another unit of velocity to the position Vector2D. Then it check s if the new position is off the screen and kills it if it goes off. If the asteroid is not alive, it “creates” a new asteroid by setting IsAlive to true and creating a random X coordinate to start the asteroid at from the top of the screen. Also, the newly created asteroid is given a velocity by using MathHelper.Lerp, which using linear interpolation, to pick a number between a minimum and maximum velocity.

### Asteroid Breakup

The asteroids in the OTV Miner game have three different sizes or levels that need to be broken up before an element can be released. All asteroids start as Large asteroids, then when hit once by a laser, the go to Medium, and then when hit again, to a Small asteroid. One more hit will cause an element to be release that the ship can collect. Since asteroids needed state information, I created a new class call AsteroidObject that inherits GameObject to add the needed state information. AsteroidObject.cs can be viewed below.

class AsteroidObject : GameObject

{

AsteroidState curState;

public AsteroidObject(Texture2D loadedTexture)

: base(loadedTexture)

{

curState = AsteroidState.Large;

}

public AsteroidState AsteroidState

{

get { return curState; }

set { curState = value; }

}

}

The asteroid state is an enum that holds the three states. Below is the enum held in the AsteroidObject file.

enum AsteroidState

{

Large,

Medium,

Small

}

The main logic behind the asteroid breakup is that it changes states after each hit (covered in Target Hitting section). Also there are three large asteroids images, three medium images, and four small images. Each time that the state changes, the program randomly assigns an image to the AsteroidObject’s Sprite. The code below is called when a collision is detected between an asteroid and a laser.

if (asteroid.AsteroidState == AsteroidState.Large)

{

asteroid.Sprite = GetRandomMediumAsteroidContent();

asteroid.AsteroidState = AsteroidState.Medium;

break;

}

if (asteroid.AsteroidState == AsteroidState.Medium)

{

asteroid.Sprite = GetRandomSmallAsteroidContent();

asteroid.AsteroidState = AsteroidState.Small;

break;

}

if (asteroid.AsteroidState == AsteroidState.Small)

{

asteroid.Sprite = GetRandomLargeAsteroidContent();

asteroid.AsteroidState = AsteroidState.Large;

GameObject element = new GameObject(GetRandomElement());

element.Position = asteroid.Position;

element.IsAlive = true;

elements.Add(element);

asteroid.IsAlive = false;

break;

}

The AsteroidState.Small state also contains code that creates an element for release and will explained in the next section.

Each GetRandom\*\*\*AsteroidContent method is listed below along with the pictures that it is creating.

private Texture2D GetRandomLargeAsteroidContent()

{

double num = (random.NextDouble() \* 100);

if (num <= 33.33)

return Content.Load<Texture2D>("Sprites\\LargeAsteroid1");

if (num > 33.33 && num <= 66.66)

return Content.Load<Texture2D>("Sprites\\LargeAsteroid2");

return Content.Load<Texture2D>("Sprites\\LargeAsteroid3");

}

|  |  |  |
| --- | --- | --- |
| LargeAsteroid1.png | LargeAsteroid2.png | LargeAsteroid3.png |
| LargeAsteroid1 | LargeAsteroid2 | LargeAsteroid3 |

private Texture2D GetRandomMediumAsteroidContent()

{

double num = (random.NextDouble() \* 100);

if (num <= 33.33)

return Content.Load<Texture2D>("Sprites\\MediumAsteroid1");

if (num > 33.33 && num <= 66.66)

return Content.Load<Texture2D>("Sprites\\MediumAsteroid2");

return Content.Load<Texture2D>("Sprites\\MediumAsteroid3");

}

|  |  |  |
| --- | --- | --- |
| MediumAsteroid1.png | MediumAsteroid2.png | MediumAsteroid3.png |
| MediumAsteroid1 | MediumAsteroid2 | MediumAsteroid3 |

private Texture2D GetRandomSmallAsteroidContent()

{

double num = (random.NextDouble() \* 100);

if (num <= 25)

return Content.Load<Texture2D>("Sprites\\SmallAsteroid1");

if (num > 25 && num <= 50)

return Content.Load<Texture2D>("Sprites\\SmallAsteroid2");

if (num > 50 && num <= 75)

return Content.Load<Texture2D>("Sprites\\SmallAsteroid3");

return Content.Load<Texture2D>("Sprites\\SmallAsteroid4");

}

|  |  |  |  |
| --- | --- | --- | --- |
| SmallAsteroid1.png | SmallAsteroid2.png | SmallAsteroid3.png | SmallAsteroid4.png |
| SmallAsteroid1 | SmallAsteroid2 | SmallAsteroid3 | SmallAsteroid4 |

### Element Release

Element Release is handled in a similar way of that of the asteroids breakup. If a Small Asteroid is hit with a laser, then the program randomly picks a element to release. There are three different types of elements that get release. Below is the function that random releases the elements and the images displayed. Currently I haven’t created the actual images for the elements so I used stand-ins for the first project. This is one of the great qualities of the XNA language, you can use any image when you update the image (with same name) the program will automatically display the new image without any code changes.

private Texture2D GetRandomElement()

{

double num = (random.NextDouble() \* 100);

if (num <= 33.33)

return Content.Load<Texture2D>("Sprites\\LaserFuel");

if (num > 33.33 && num <= 66.66)

return Content.Load<Texture2D>("Sprites\\PropulsionFuel");

return Content.Load<Texture2D>("Sprites\\OsmiumOre");

}

|  |  |  |
| --- | --- | --- |
| LaserFuel.png | PropulsionFuel.png | OsmiumOre.png |
| LaserFuel (Green) | PropulsionFuel (Blue) | OsmiumOre (White) |

Once the element is released from a small asteroid, it has no velocity and doesn’t move. The element will be collected when the ship “collides” with the element and then it will not be displayed any longer. Therefore, I created a separate collection of elements in the program that grows based on the number of elements that have been mined but not collected.

### Target Hitting

Target Hitting is handled in this project by determining if the two objects’ rectangles in question intersect each other. I am not happy with this implementation for the final product and will update the collision detection algorithm before the final product. The only collisions detected are that of a laser and an asteroid. This detection occurs in the UpdateLasers() function in the Game1 class.

private void UpdateLasers()

{

foreach (GameObject laser in lasers)

{

if (laser.IsAlive)

{

laser.Position += laser.Velocity;

if (!viewportRect.Contains(new Point((int)laser.XPos,

(int)laser.YPos)))

{

laser.IsAlive = false;

continue;

}

Rectangle laserRect = new Rectangle(

(int)laser.XPos,

(int)laser.YPos,

laser.Sprite.Width,

laser.Sprite.Height);

foreach (AsteroidObject asteroid in asteroids)

{

Rectangle asteroidRect = new Rectangle(

(int)asteroid.XPos,

(int)asteroid.YPos,

asteroid.Sprite.Width,

asteroid.Sprite.Height);

if (laserRect.Intersects(asteroidRect))

{

laser.IsAlive = false;

if (asteroid.AsteroidState == AsteroidState.Large)

{

asteroid.Sprite =

GetRandomMediumAsteroidContent();

asteroid.AsteroidState = AsteroidState.Medium;

break;

}

if (asteroid.AsteroidState == AsteroidState.Medium)

{

asteroid.Sprite = GetRandomSmallAsteroidContent();

asteroid.AsteroidState = AsteroidState.Small;

break;

}

if (asteroid.AsteroidState == AsteroidState.Small)

{

asteroid.Sprite = GetRandomLargeAsteroidContent();

asteroid.AsteroidState = AsteroidState.Large;

GameObject element = new

GameObject(GetRandomElement());

element.Position = asteroid.Position;

element.IsAlive = true;

elements.Add(element);

asteroid.IsAlive = false;

break;

}

}

}

}

}

The highlighted code in the above code determines if there was a collsion between a laser and an asteroid. I plan to do more research on collision detection and come up with a better detection.

In order to have a collision between a laser and an asteroid, we first need to fire a laser. This was handled in the Update function if the spacebar was pressed. Below this the code actually fires the laser. The laser only fires if there is a available “dead” laser in the collection to fire. It also sets the position the middle of ship and gets it a initial velocity.

private void FireLasers()

{

foreach (GameObject laser in lasers)

{

if (!laser.IsAlive)

{

laser.IsAlive = true;

laser.Position = theShip.Position + theShip.Center;

laser.Velocity = new Vector2(0.0f, -5.0f);

return;

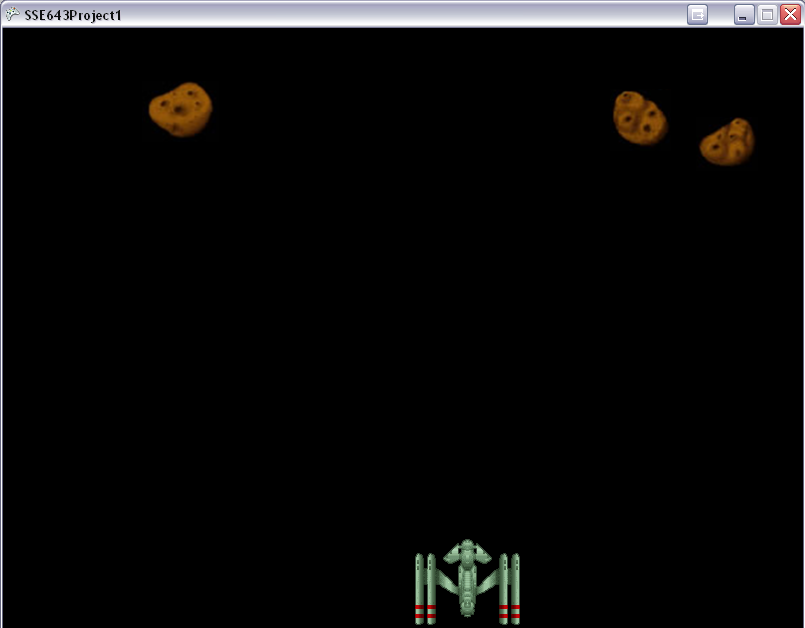
}

}

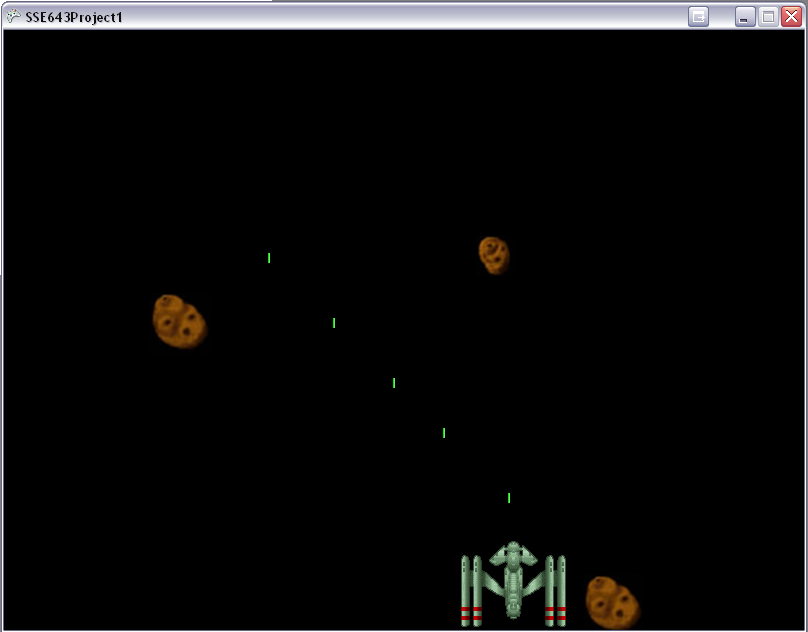
}

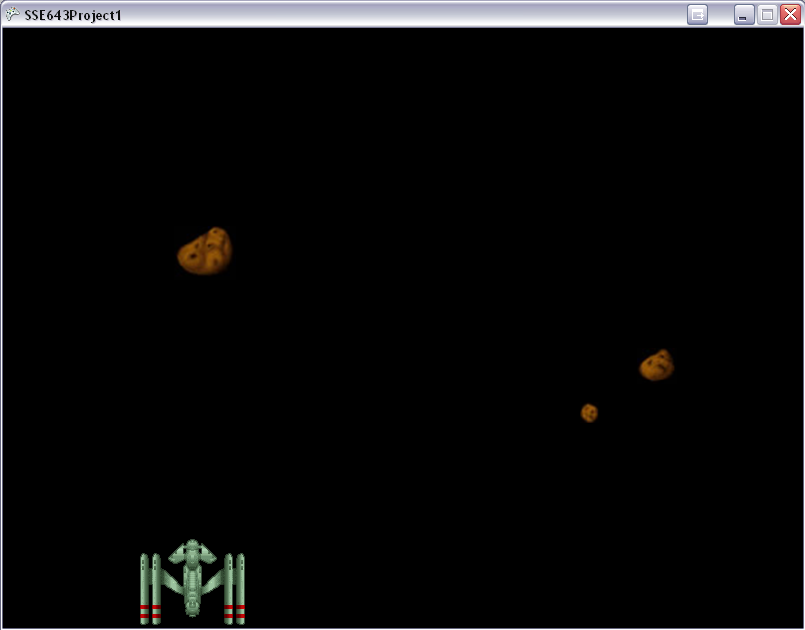
### Gameplay

The gameplay in Project 1 will not be the final product but was created to demonstrate the goals of this project. When the game is started, the ship is at the bottom of screen and three (which is the max amount of asteroids) start “falling” from the top of the screen.

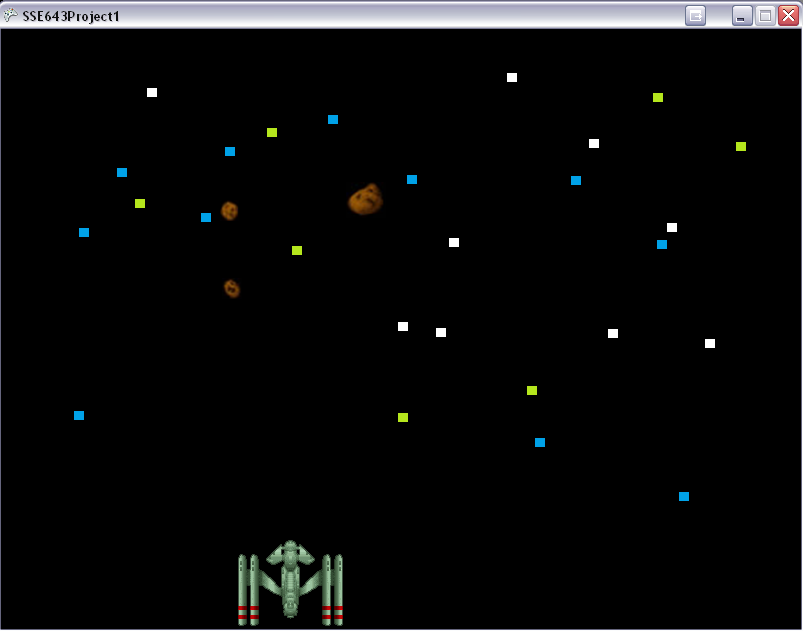


Once the spacebar is pressed a laser is fired and it follows a straight line trajectory from where the ship was when the spacebar was pressed.



The next screenshot depicts the three different states of the asteroids on the screen. From left to right: Large, Small, and Medium.

The last screenshot was taken after several asteroids have been broken down and elements have been released.



## Conclusion

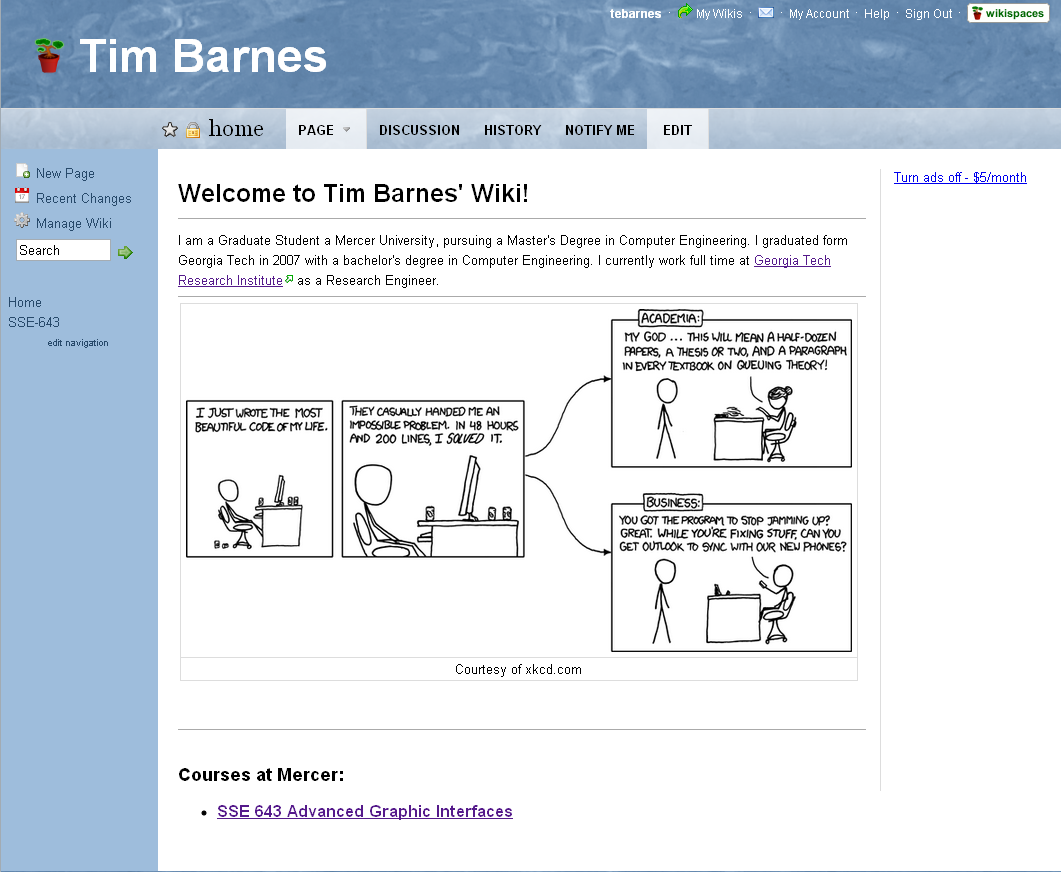
The XNA framework has made it relatively easy to create, draw, and move objects on the screen by utilizing the LoadContent, Update, and Draw methods. I have created games before just using the standard C# .NET libraries and the management of each object and drawing was time consuming process that interfered with the creation of the game. With XNA, most of the “bookkeeping” is keep to a minimum and the programmer is able to focus on creation and animating objects. I am very new at XNA but I can already see its power in creating games and I hope learn a lot more throughout this course.

# Part 2: Wiki

The second part of Project 1 was to learn how to create and edit a wiki page(s) for this class. I have used many wiki pages for work but the wiki we are using in class is more of a hybrid webpage that has a text editor that updates the page on a save. This took some work to learn and figure out.

## Main Page

I wanted to create a “Welcome” page that would hold links to other pages for each class I will be taking while at Mercer. I added a little about me section to the top, then added a funny picture that I have had for while about programmers, and finally added a links section that links to another page for each class that will be held on the wiki.

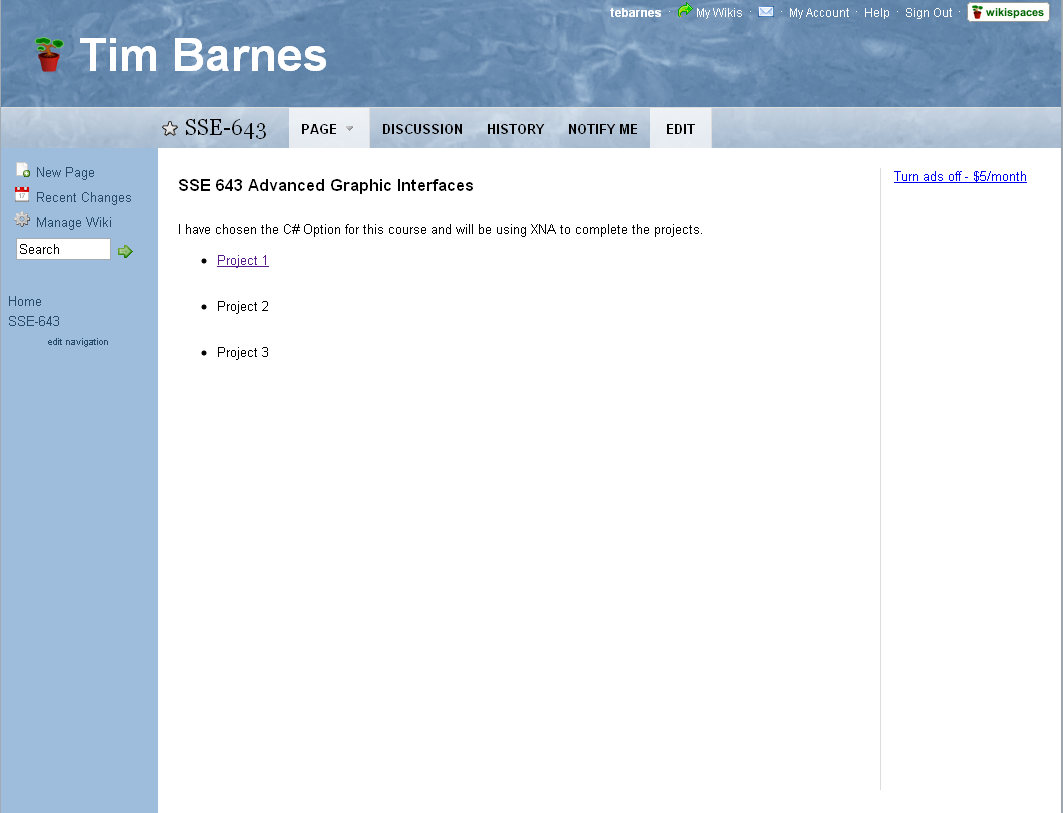


This was all done by simply clicking the Edit button on the top toolbar and typing in the text and importing the picture from my hard drive and uploading it to the website. Once I created the SSE643 page that will hold all of my projects from this class, I was able to highlight the “SEE 643 Advanced Graphic Interfaces” text and link it to the SSE643 page. This created a hyperlink to the newly created page.

I also didn’t like the default the template of the wiki page so I change it to the blue “glassy” look that it is now. I really don’t the plant flower box icon, but haven’t found another image that I want to display, I will continue to look for one and maybe have it up before I call this project finished.

## SSE 643 Advanced Graphics Interfaces Page

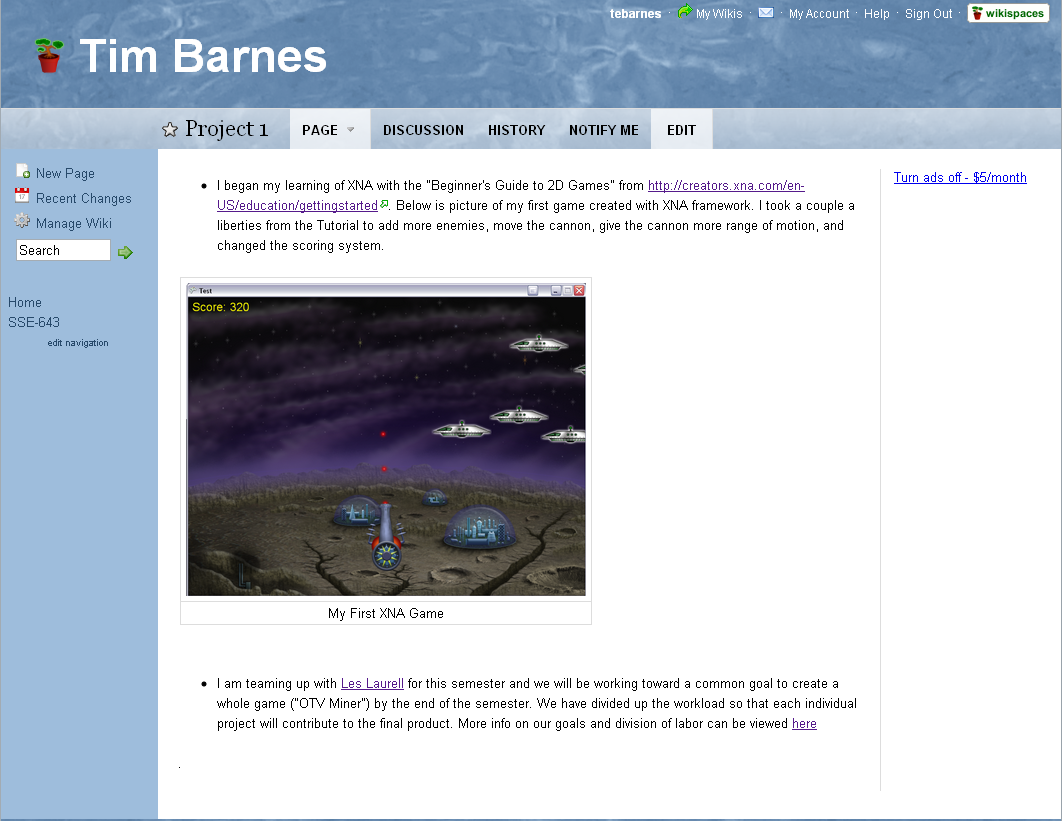
Similar to the main page, I created this page by clicking the Edit button and added the desired text. This page will contain links to separate pages for each project in this class. Currently, I have only done Project 1, so that is the only one with a hyperlink.



## Project 1 Page

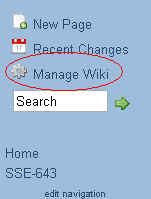
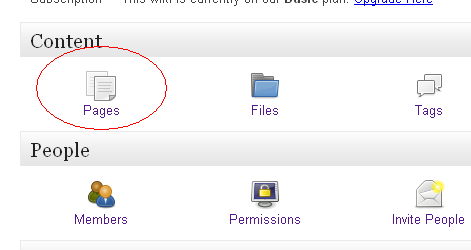
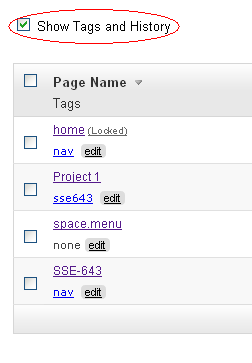
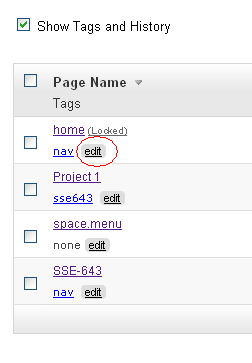
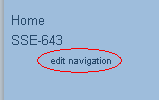
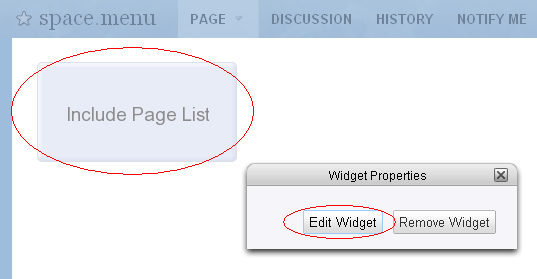
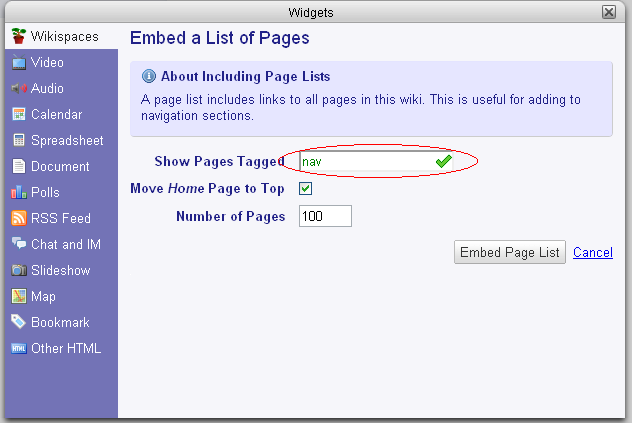
Project 1 page, which at the time of this report is not completed, contains a brief summary of the full project report. On this page, I have a brief description and link to the “Beginner’s Guide to 2D Games.” I also added a screenshot of my first game. Loaded pictures and added hyperlinks are mere clicks of a mouse. I also added a link to my partner’s Wiki page (Les Laurell) and a link to our semester project overview that is held on his wiki page. I didn’t see a new to upload the same file to the server when I could just simply add a hyperlink.

When this Project 1 report is done, I plan to post the Word document to this wiki page.



## Cleaning the Navigation Bar

During the creation of several wiki pages, my navigation bar got cluttered with pages I didn’t want to display. After playing around for awhile I found a way to remove unwanted pages to be displayed on the navigation bar. I will walk you through the process in this section.

* First, click Manage Wiki from the Navigation Bar.
  + 
* Click on Pages, under the Content Header
  + 
* Check the “Show Tags and History”
  + 
* Next click edit on a page that you want ON the Navigation bar
  + 
* Type in a Tag
* Repeat on all pages that you want on the navigation bar
* When done, click on “edit navigation” on the Navigation Bar
  + 
* Click on the “Include Page List” and then Click “Edit Widget”
  + 
* Now type the same Tag in to the “Show Pages Tagged” text box
  + 
* Close widget and press Save on the editor toolbar. Notice that the Navigation Bar is now filtered.